УДК 519.7

Г. Г. СЕРГЕЕВ, канд. техн. наук, доц. СевНТУ, г. Севастополь

ФОРМАЛИЗАЦИЯ ПРОЦЕССА РЕФАКТОРИНГА НА ОСНОВЕ СИМВОЛЬНОЙ ЗАПИСИ СТРУКТУРЫ КЛАССОВ

В статті пропонується символьний запис структури класів програмної системи як засобу абстрактного синтаксису та семантики понять об'єктного моделювання. Розглядаються операції над символьним записом і їх вживання в процесі оптимізації програмних систем.

В статье предлагается символьная запись структуры классов программной системы как средства абстрактного синтаксиса и семантики понятий объектного моделирования. Рассматриваются операции над символьной записью и их применение в процессе рефакторинга программных систем.

In this paper symbol record of class structure is offered. It is tools abstract syntax and semantics where objective design used. Operations above a symbol record are declaring. Process of code refactoring are supported by.

Введение. В настоящие время практически все программное обеспечение использует средства объектно-ориентированного программирования. Именно это объясняет исключительную важность задач, связанных с облегчением сопровождения и развития существующего программного кода [1]. В то же время, этим задачам уделяется недостаточное внимание, ощущается явный недостаток методик и эффективных инструментов поддержки работы с существующим кодом. При разработке объектно-ориентированных систем, необходимо решать две классических задачи: задачу анализа и задачу синтеза. Решение задач синтеза во многом автоматизировано средствами различных CASE систем, Framework и т.п. Для моделирования бизнес-процессов системного проектирования и отображения организационных структур в большинстве приложений используется язык UML. В работах [2, 3] рассматривается использование UML-диаграмм для решения задачи синтеза и декларативные подходы к решению задачи анализа. Однако решение задачи анализа на основе UML не формализовано, а сама диаграмма не дает возможности выполнять формальные преобразования над классами с целью рефакторинга проекта [4].

Формальное представление структуры классов программных объектов в виде символьной записи. На основании материалов работы [2], можно утверждать, что класс – это триплет вида $C = \{Pv, Pt, Pb\}$, где Pv, Pt, Pb – множество частных, защищенных и публичных данных и методов класса. С другой стороны, основываясь на свойстве инкапсуляции, класс – это пара $C = \{F, M\}$, где F – множество полей, а M – множество методов класса. Таким образом,

$$C = \{PvF \bigcup PvM, PtF \bigcup PtM, PbF \bigcup PbM\} \quad (1)$$

где PvF, PtF, PbF - множество частных, защищенных и публичных полей класса, PvM, PtM, PbM - множество частных, защищенных и публичных методов класса. Очевидно, что $PvF \in F, PtF \in F, PbF \in F$, $PvM \in M, PtM \in M, PbM \in M$, $PvF \bigcap PtF = \varnothing$, $PvF \bigcap PbF = \varnothing$, $PtF \bigcap PbF = \varnothing$, $PvM \bigcap PtM = \varnothing$, $PvM \bigcap PbM = \varnothing$, $PtM \bigcap PbM = \varnothing$, $PvF \bigcup PtF \bigcup PbF = F$, $PvM \bigcup PtM \bigcup PbM = M$.

Для описания структуры класса в виде символической записи предлагается следующая нотация. Символ, обозначающий атрибут класса, записывается как N^{type}, где N - имя поля, а $type$ – тип поля. Метод класса предлагается записывать как $Meth^{type,}_{prim}(param)$, где $Meth$ – идентификатор метода, $param$ – перечень формальных параметров, а $prim \subseteq [virtual, abstarct, override]$. Допустимо, что $prim = \varnothing$ и/или $param = \varnothing$. Класс определяется как $Cname_{Cprim}$, где $Cprim = abstarct \left| \varnothing \right.$. Синтаксис символьной записи структуры класса определяется грамматикой, представленной в таблице 1.

Таблица 1 – Грамматика символьной записи структуры класса

<ClassDeclare>→<CN>=<CNbase>+{[<PrvF>,<PrvM>];[<PrtF>,<PrtM>]; [<PubF>,<PubM>]}	
<CN>→<id><Cprim>	<FIELD>→<id><type>
< PrvF>→<FIELDS>	<FIELD>→ε
< PrtF>→<FIELDS>	<METHODS>→<METHOD>
<PubF>→<FIELDS> <FIELDS>→ <FIELD>	<METHODS>→<METHOD>, <METHODS>
< PrvM >→<METHODS>	<METHOD>→<id><type><prim><param>()
< PrtM >→<METHODS>	<Cprim>→abstarct|ε
< PubM >→<METHODS>	<prim>→ virtual|abstract|override|ε
<FIELDS>→|<FIELD>,<FIELDS>	<CNbase>→<id>+|ε

Таким образом, в символьном виде описание нового класса *newClass*, порожденного от базового класса *baseCalss* это выражение вида

$$newClass = baseClass + \{[PvF, PvM], [PtF, PtM], [PbF, PbM]\}$$

Рассмотрим пример символьной записи структуры классов, показанной в виде UML-диаграммы на рис. 1.

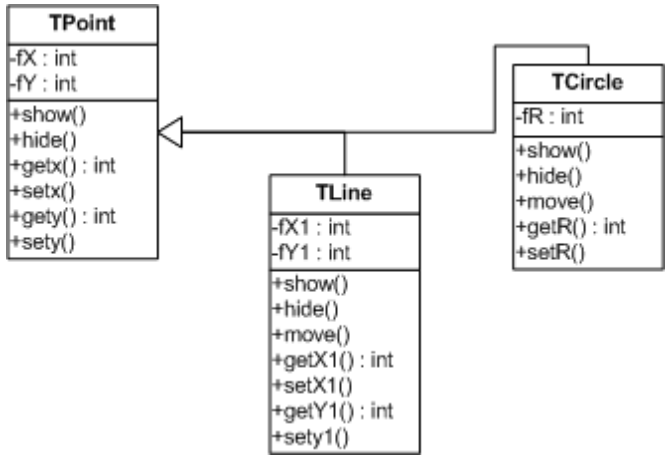

Рис. 1 – UML-диаграмма представления классов геометрических фигур

$$TPoint = \begin{Bmatrix} [fX^{\text{int}}, fY^{\text{int}}],[],[show_{virtual}(), hide_{virtual}(), getX^{\text{int}}_{virtual}(), \\ setX_{virtual}(newX^{\text{int}}), getY^{\text{int}}_{virtual}(), setY_{virtual}(newY^{\text{int}})] \end{Bmatrix},$$

$$TLine = Tpoint + \begin{Bmatrix} [fX1^{\text{int}}, fY1^{\text{int}}],[],[show_{override}(), hide_{override}(), \\ getX1^{\text{int}}_{virtual}(), setX1_{virtual}(newX1^{\text{int}}), getY1^{\text{int}}_{virtual}(), \\ setY1_{virtual}(newY1^{\text{int}})] \end{Bmatrix},$$

$$TCircle = Tpoint + \begin{Bmatrix} [fR^{\text{int}}],[],[show_{override}(), hide_{override}(), getR^{\text{int}}_{virtual}(), \\ setR_{virtual}(newR^{\text{int}})] \end{Bmatrix}.$$

Очевидно, что символьная запись структуры классов компактнее соответствующей ей UML-диаграммы. Определим на основании предложенной символьной записи операции над классами для решения задачи анализа структуры проекта и его последующего рефакторинга.

Определим для двух произвольных классов $C1$ и $C2$ операцию пересечения, которую будем записывать как $C1 \cap C2$. Введем в рассмотрение функцию $I(\chi_1, \chi_2) = [\chi_1 \setminus (\chi_1 \cap \chi_2), \chi_2 \setminus (\chi_1 \cap \chi_2)]$. Пусть операция $C1 \cap C2$. в соответствии со спецификацией (1) определяется как:

$$baseC = \begin{Bmatrix} [PvF_{C1} \cap PvF_{C2}, PvM_{C1} \cap PvM_{C2}], \\ [PtF_{C1} \cap PtF_{C2}, PtM_{C1} \cap PtM_{C2}], \\ [PbF_{C1} \cap PbF_{C2}, PbM_{C1} \cap PbM_{C2}] \end{Bmatrix},$$

$$C1 \cap C2 = baseC + \begin{Bmatrix} [I(PvF_{C1}, PvF_{C2}), I(PvM_{C1}, PvM_{C2})], \\ [I(PtF_{C1}, PtF_{C2}), I(PtM_{C1}, PtM_{C2})], \\ [I(PbF_{C1}, PbF_{C2}), I(PbM_{C1}, PbM_{C2})] \end{Bmatrix}.$$

Базовый класс *baseC* – это ближайший общий предок в дереве иерархии или единый базовый класс *BaseObject* (NSObject, java.lang.Object, System.Object, TObject и т. п.), если такого не существует. Если реализация методов с одинаковым объявлением PvM, PtM, PbM различна для операндов операции пересечения, то в результате пересечения они получают дополнительный префикс *abstract*. Проиллюстрируем операцию строгого пересечения на примере UML-диаграммы классов, представленных на рис. 2.

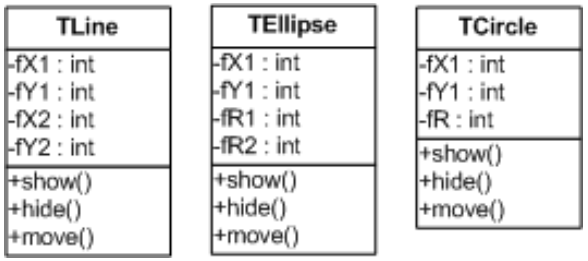

Рис. 2 – UML-диаграмма классов без общего предка

$$TEllipse \cap TLine = BaseObject + \begin{Bmatrix} [fX1^{\text{int}}, fY1^{\text{int}}],[\,], \\ [show_{virtua,abstractl}(), \\ hide_{virtual,abstract}(), move()] \end{Bmatrix},$$

$$TEllipse \cap TCircle = TEllipse \cap TLine,$$

$$TCircle \cap TCircle = TEllipse \cap TLine.$$

Результатом пересечения классов, как правило, является новый класс или общий предок классов, заявленных в качестве операндов. Таким же образом определяем операцию вычитания классов $C1 \setminus C2$:

$$C1 \setminus C2 = \begin{Bmatrix} [PvF_{C1} \setminus PvF_{C2}, PvM_{C1} \setminus PvM_{C2} + \underset{Mi \in C1}{\forall} Mi_{abstract}], \\ [PtF_{C1} \setminus PtF_{C2}, PtM_{C1} \setminus PtM_{C2} + \underset{Mi \in C1}{\forall} Mi_{abstract}], \\ [PbF_{C1} \setminus PbF_{C2}, PbM_{C1} \setminus PbM_{C2} + \underset{Mi \in C1}{\forall} Mi_{abstract}] \end{Bmatrix}.$$

То есть операция вычитания есть традиционное вычитание множеств полей и методов класса с учетом необходимости перегрузки абстрактных методов.

Решение задачи рефакторинга на основе символьной записи. Образование новых классов в результате выполнения операции пересечения является основой для процесса рефакторинга, описанного в [3]. При этом следует действовать по следующему формальному алгоритму вне зависимости от лингвистических сред и особенностей реализации.

Шаг 1. Выполнить преобразование в символьную запись всех объявлений классов программной системы.

Шаг 2. Для всех классов, имеющих общего предка в дереве иерархии выполняется операция строгого пересечения: $TParentObject = \bigcap_{i}^{i} C_i$.

Шаг 3. Если результат операции пересечения не равен *TParentObject≠BaseObject*, определяем новый класс предка.

Шаг 4. Выполняем рефакторинг кода путем вычитания из каждого класса C_i класса TParentObject.

Рассмотрим пример рефакторинга программных классов, представленных на рисунке 2.

$$TFigure = BaseObject + \left\{ \begin{matrix} [fX1^{\text{int}}, fY1^{\text{int}}],[\ ], \\ [show_{virtua,abstractl}(), hide_{virtual,abstract}(), move()] \end{matrix} \right\},$$

$$TLine = TFigure + \{[fX2^{\text{int}}, fY2^{\text{int}}],[],[show_{override}(), hide_{override}()]\},$$

$$TEllipse = TFigure + \{[fR1^{\text{int}}, fR2^{\text{int}}],[],[show_{override}(), hide_{override}()]\},$$

$$TCircle = TFigure + \{[fR^{\text{int}}],[],[show_{override}(), hide_{override}()]\}.$$

Выводы. Формальный подход к решению задачи рефакторинга программных систем на основе символьной записи структуры классов является основой для построения программного продукта, позволяющего выполнять автоматизированный рефакторинг архитектуры программных систем практически без участия программиста.

Список литературы: **1.** *Ксензов М. В.* Рефакторинг архитектуры программного обеспечения / *М. В. Ксензов* // Труды института системного программирования РАН. – 2004. – Т. 8. – С. 180–192. **2.** *Буч Г.* Язык UML. Руководство пользователя / *Д. Рамбо, А. Джеконсон*. – СПб. : ДМК Пресс, 2007. – 435 с. **3.** *Буч Г.* Объектно-ориентированный анализ и проектирование приложений на C++ / *Г. Буч*. – М. : Бином, 1998. – 560 с. **4.** *Фаулер М.* Рефакторинг: улучшение существующего кода / *М. Фаулер*. – СПб. : Символ, 2003. – 432 с.

Надійшла до редколегії 13.11.2010